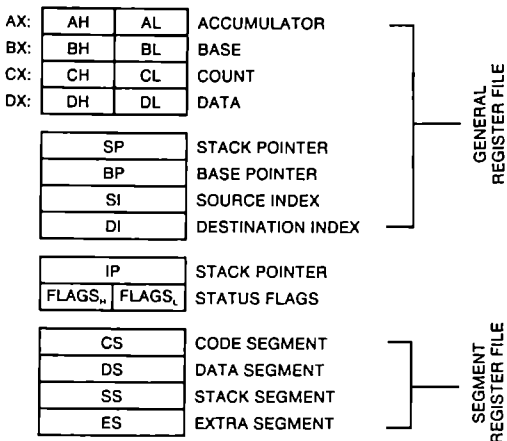

ASM86 MACRO ASSEMBLER POCKET REFERENCE

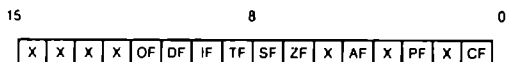
CONTENTS

	PAGE
8086 Register Model	1
Operand Summary	2
Second Instruction Byte Summary	2
Operand Address (EA) Timing (Clocks)	2
Memory Segmentation Model	3
8086/8088 Instructions	4
186 Instructions	36
8087 Instructions	41
Assembler Controls Summary	71
ASM86 Invocation	73
Assembler Directives	74
Processor Reset Register Initialization	75
MCS-86 Reserved Locations	75
iAPX 86/88/186 Instruction Set Matrix	77
Clocks for MOD186 Operation	79

8086 Register Model



Instructions that reference the flag register file as a 16-bit object use the symbol **FLAGS** to represent the file:



X = Don't Care

Flags

- AF: AUXILIARY CARRY — BCD
- CF: CARRY FLAG
- DF: DIRECTION FLAG (STRINGS)
- IF: INTERRUPT ENABLE FLAG
- OF: OVERFLOW FLAG (CF SF)
- PF: PARITY FLAG
- SF: SIGN FLAG
- TF: TRAP (SINGLE STEP FLAG)
- ZF: ZERO FLAG

Operand Summary

"reg" field Bit Assignments:

Word Operand	Byte Operand	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 OS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Second Instruction Byte Summary

mod xxx r/m

mod	Displacement
00	DISP = 0*, disp-low and disp-high are absent
01	DISP = disp-low sign-extended to 16-bits, disp-high is absent
10	DISP = disp-high: disp-low
11	r/m is treated as a "reg" field

r/m	Operand Address
000	(BX) + (SI) + DISP
001	(BX) + (DI) + DISP
010	(BP) + (SI) + DISP
011	(BP) + (DI) + DISP
100	(SI) + DISP
101	(DI) + DISP
110	(BP) + DISP*
111	(BX) + DISP

DISP follows 2nd byte of instruction (before data if required).

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low

Operand Address (EA) Timing (Clocks):

Add 4 clocks for word operands at ODD ADDRESSES.

Immed Offset = 6

Base (BX, BP, SI, DI) = 5

Base + DISP = 9

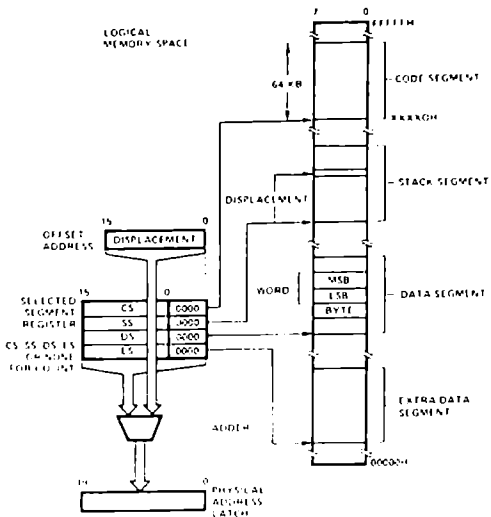
Base + Index (BP + DI, BX + SI) = 7

Base + Index (BP + SI, BX + DI) = 8

Base + Index (BP + DI, BX + SI) + DISP = 11

Base + Index (BP + SI, BX + DI) + DISP = 12

Memory Segmentation Model



Segment Override Prefix

0 0 1 reg 1 1 0

Timing: 2 clocks

Use of Segment Override

Operand Register	Default	With Override Prefix
IP (code address)	CS	Never
SP (stack address)	SS	Never
BP (stack address or stack marker)	SS	BP → DS or ES, or CS
SI or DI (not incl. strings)	DS	ES, SS, or CS
SI (implicit source addr for strings)	DS	ES, SS, or CS
DI (implicit dest addr for strings)	ES	Never

8086/8088 Instructions

Notes for 8086/8088 Instructions

The individual instruction descriptions are shown by a format box such as the following:

Opcode	m/op/r/m					Data		
--------	----------	--	--	--	--	------	--	--

These are byte-wise representations of the object code generated by the assembler and are interpreted as follows:

- Opcode is the 8-bit opcode for the instruction. The actual opcode generated is defined in the "Opcode" column of the instruction table that follows each format box.
- m/op/r/m is the byte that specifies the operands of the instruction. It contains a 2-bit mode field (m), a 3-bit register field (op), and a 3-bit register or memory (r/m) field.
- Dashed blank boxes following the m/op/r/m box are for any displacement required by the mode field.
- Data is for a byte of immediate data.
- A dashed blank box following a Data box is used whenever the immediate operand is a word quantity.

AAA = ASCII Adjust for Addition

Opcode

Opcode	Clocks	Operation
37	4	adjust AL, flags, AH

AAD = ASCII Adjust for Division

Long—Opcode

Opcode	Clocks	Operation
D5,0A	60	Adjust AL, AH prior to division

AAM = ASCII Adjust for Multiplication

Long—Opcode

Opcode	Clocks	Operation
D4,0A	83	Adjust AL, AH after multiplication

AAS = ASCII Adjust for Subtraction

Opcode

Opcode	Clocks	Operation
3F	4	adjust AL, flags, AH

ADC = Integer Add with Carry

Memory/Reg + Reg

Opcode	mod reg r/m				
--------	-------------	--	--	--	--

	Opcode	Clocks	Operation
Byte	12	3	$\text{Reg8} \leftarrow \text{CF} + \text{Reg8} + \text{Reg8}$
	12	$9 + \text{EA}$	$\text{Reg8} \leftarrow \text{CF} + \text{Reg8} + \text{Mem8}$
	10	$16 + \text{EA}$	$\text{Mem8} \leftarrow \text{CF} + \text{Mem8} + \text{Reg8}$
Word	13	3	$\text{Reg16} \leftarrow \text{CF} + \text{Reg16} + \text{Reg16}$
	13	$9 + \text{EA}$	$\text{Reg16} \leftarrow \text{CF} + \text{Reg16} + \text{Mem16}$
	11	$16 + \text{EA}$	$\text{Mem16} \leftarrow \text{CF} + \text{Mem16} + \text{Reg16}$

Immed to AX/AL

Opcode	Data		
--------	------	--	--

	Opcode	Clocks	Operation
Byte	14	4	$\text{AL} \leftarrow \text{CF} + \text{AL} + \text{Immed8}$
Word	15	4	$\text{AX} \leftarrow \text{CF} + \text{AX} + \text{Immed16}$

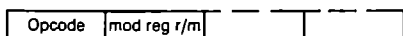
Immed to Memory/Reg

Opcode	mod 010 r/m					Data		
--------	-------------	--	--	--	--	------	--	--

	Opcode	Clocks	Operation
Byte	80	4	$\text{Reg8} \leftarrow \text{CF} + \text{Reg8} + \text{Immed8}$
	80	$17 + \text{EA}$	$\text{Mem8} \leftarrow \text{CF} + \text{Mem8} + \text{Immed8}$
Word	81	4	$\text{Reg16} \leftarrow \text{CF} + \text{Reg16} + \text{Immed16}$
	81	$17 + \text{EA}$	$\text{Mem16} \leftarrow \text{CF} + \text{Mem16} + \text{Immed16}$
	83	4	$\text{Reg16} \leftarrow \text{CF} + \text{Reg16} + \text{Immed8}$
	83	$17 + \text{EA}$	$\text{Mem16} \leftarrow \text{CF} + \text{Mem16} + \text{Immed8}$

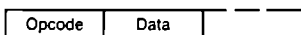
ADD = Integer Addition

Memory/Reg + Reg



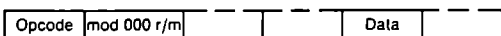
	Opcode	Clocks	Operation
Byte	02	3	$\text{Reg8} \leftarrow \text{Reg8} + \text{Reg8}$
	02	9+EA	$\text{Reg8} \leftarrow \text{Reg8} + \text{Mem8}$
	00	16+EA	$\text{Mem8} \leftarrow \text{Mem8} + \text{Reg8}$
Word	03	3	$\text{Reg16} \leftarrow \text{Reg16} + \text{Reg16}$
	03	9+EA	$\text{Reg16} \leftarrow \text{Reg16} + \text{Mem16}$
	01	16+EA	$\text{Mem16} \leftarrow \text{Mem16} + \text{Reg16}$

Immed to AX/AL



Opcode	Clocks	Operation
04	4	$\text{AL} \leftarrow \text{AL} + \text{Immed8}$
05	4	$\text{AX} \leftarrow \text{AX} + \text{Immed16}$

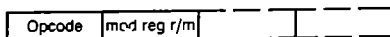
Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	80	4	$\text{Reg8} \leftarrow \text{Reg8} + \text{Immed8}$
	80	17+EA	$\text{Mem8} \leftarrow \text{Mem8} + \text{Immed8}$
Word	81	4	$\text{Reg16} \leftarrow \text{Reg16} + \text{Immed16}$
	81	17+EA	$\text{Mem16} \leftarrow \text{Mem16} + \text{Immed16}$
	83	4	$\text{Reg16} \leftarrow \text{Reg16} + \text{Immed8}$
	83	17+EA	$\text{Mem16} \leftarrow \text{Mem16} + \text{Immed8}$

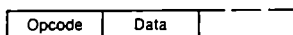
AND = Logical AND

Memory/Reg with Reg



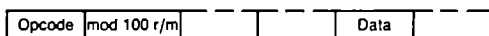
	Opcode	Clocks	Operation
Byte	22	3	Reg8 ← Reg8 AND Reg8
	22	9 + EA	Reg8 ← Reg8 AND Mem8
	20	16 + EA	Mem8 ← Mem8 AND Reg8
Word	23	3	Reg16 ← Reg16 AND Reg16
	23	9 + EA	Reg16 ← Reg16 AND Mem16
	21	16 + EA	Mem16 ← Mem16 AND Reg16

Immed to AX/AL



	Opcode	Clocks	Operation
Byte	24	4	AL ← AL AND Immed8
Word	25	4	AX ← AX AND Immed16

Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 AND Immed8
	80	17 + EA	Mem8 ← Mem8 AND Immed8
Word	81	4	Reg16 ← Reg16 AND Immed16
	81	17 + EA	Mem16 ← Mem16 AND Immed16

CALL = Call

Within segment or group, IP relative

Opcode	DispL	DispH
--------	-------	-------

Opcode	Clocks	Operation
E8	19	$IP \leftarrow IP + \text{Disp16}$ —(SP) $\leftarrow$ return link

Within segment or group, Indirect

Opcode	mod 010 r/m				
--------	-------------	--	--	--	--

Opcode	Clocks	Operation
FF	16	$IP \leftarrow \text{Reg16}$ —(SP) $\leftarrow$ return link
FF	21 + EA	$IP \leftarrow \text{Mem16}$ —(SP) $\leftarrow$ return link
FF	21 + EA	$IP \leftarrow \text{Mem16}$ —(SP) $\leftarrow$ return link

Inter-segment or group, Direct

Opcode	offset	offset	segbase	segbase	segbase
--------	--------	--------	---------	---------	---------

Opcode	Clocks	Operation
9A	28	$CS \leftarrow \text{segbase}$ $IP \leftarrow \text{offset}$

Inter-segment or group, Indirect

Opcode	mod 011 r/m				
--------	-------------	--	--	--	--

Opcode	Clocks	Operation
FF	37 + EA	$CS \leftarrow \text{segbase}$ $IP \leftarrow \text{offset}$

CBW = Convert Byte to Word

Opcode

Opcode	Clocks	Operation
98	2	convert byte in AL to word in AX

CLC = Clear Carry Flag

Opcode

Opcode	Clocks	Operation
F8	2	clear the carry flag

CLD = Clear Direction Flag

Opcode

Opcode	Clocks	Operation
FC	2	clear direction flag

CLI = Clear Interrupt Enable Flag

Opcode	Clocks	Operation
FA	2	clear interrupt flag

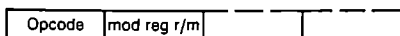
CMC = Complement Carry Flag

Opcode

Opcode	Clocks	Operation
F5	2	complement carry flag

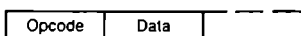
CMP = Compare Two Operands

Memory/Reg with Reg



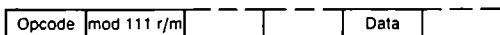
	Opcode	Clocks	Operation
Byte	38	3	flags ← Reg8 - Reg8
	38	9 + EA	flags ← Reg8 - Mem8
	3A	9 + EA	flags ← Mem8 - Reg8
Word	39	3	flags ← Reg16 - Reg16
	39	9 + EA	flags ← Reg16 - Mem16
	3B	9 + EA	flags ← Mem16 - Reg16

Immed to AX/AL



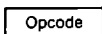
	Opcode	Clocks	Operation
Byte	3C	4	flags AL - Immed8
Word	3D	4	flags AX - Immed16

Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	80	4	flags ← Reg8 - Immed8
	80	10 + EA	flags ← Mem8 - Immed8
Word	81	4	flags ← Reg16 - Immed16
	81	10 + EA	flags ← Mem16 - Immed16
	83	4	flags ← Reg16 - Immed8
	83	10 + EA	flags ← Mem16 - Immed8

CWD = Convert Word to Doubleword



Opcode	Clocks	Operation
99	5	convert word in AX to doubleword in DX:AX

DAA = Decimal Adjust for Addition

Opcode

Opcode	Clocks	Operation
27	4	adjust AL, flags, AH

DAS = Decimal Adjust for Subtraction

Opcode

Opcode	Clocks	Operation
2F	4	adjust AL, flags, AH

DEC = Decrement by 1

Word Register

Opcode + reg

Opcode	Clocks	Operation
4B + reg	2	Reg16 $\leftarrow$ Reg16 - 1

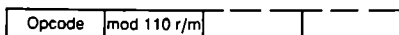
Memory/Byte Register

Opcode	mod 001 r/m				
--------	-------------	--	--	--	--

	Opcode	Clocks	Operation
Byte	FE	3	Reg8 $\leftarrow$ Reg8 - 1
	FE	15 + EA	Mem8 $\leftarrow$ Mem8 - 1
Word	FF	15 + EA	Mem16 $\leftarrow$ Mem16 - 1

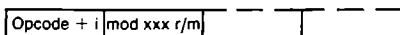
DIV = Unsigned Division

Memory/Reg with AX or DX:AX



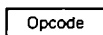
	Opcode	Clocks	Operation
Byte	F6	80-90	AH,AL $\leftarrow$ AX / Reg8
	F6	(86-96)+EA	AH,AL $\leftarrow$ AX / Mem8
Word	F7	144-162	DX,AX $\leftarrow$ DX:AX / Reg16
	F7	(150-168)+EA	DX,AX $\leftarrow$ DX:AX / Mem16

ESC = Escape



Opcode	Clocks	Operation
D8+i	8+EA	data bus $\leftrightarrow$ (EA)
D8+i	2	data bus $\leftrightarrow$ (EA)

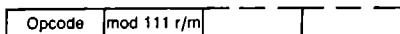
HLT = Halt



Opcode	Clocks	Operation
F4	2	halt operation

IDIV = Signed Division

Memory/Reg with AX or DX:AX



	Opcode	Clocks	Operation
Byte	F6	101-112	AH,AL $\leftarrow$ AX / Reg8
	F6	(107-118)+EA	AH,AL $\leftarrow$ AX / Mem8
Word	F7	165-184	DX,AX $\leftarrow$ DX:AX / Reg16
	F7	(171-190)+EA	DX,AX $\leftarrow$ DX:AX / Mem16

IMUL = Signed Multiplication

Memory/Reg with AL or AX

Opcode	mod 101 r/m				
--------	-------------	--	--	--	--

	Opcode	Clocks	Operation
Byte	F6	80-98	AX $\leftarrow$ AL*Reg8
	F6	(86-104)+EA	AX $\leftarrow$ AL*Mem8
Word	F7	128-154	DX:AX $\leftarrow$ AX*Reg16
	F7	(134-160)+EA	DX:AX $\leftarrow$ AX*Mem16

IN = Input Byte, Word

Fixed port

Opcode	Port
--------	------

	Opcode	Clocks	Operation
Byte	E4	10	AL $\leftarrow$ Port8
	E5	10	AX $\leftarrow$ Port8

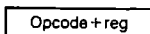
Variable port

Opcode

	Opcode	Clocks	Operation
Word	EC	8	AL $\leftarrow$ Port16(in DX)
	ED	8	AX $\leftarrow$ Port16(in DX)

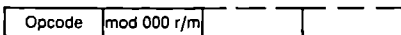
INC = Increment by 1

Word Register



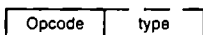
Opcode	Clocks	Operation
40+reg	2	Reg16 $\leftarrow$ Reg16 + 1

Memory/Byte Register



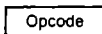
	Opcode	Clocks	Operation
Byte	FE	3	Reg8 $\leftarrow$ Reg8 + 1
	FE	15+EA	Mem8 $\leftarrow$ Mem8 + 1
Word	FF	15+EA	Mem16 $\leftarrow$ Mem16 + 1

INT INTO = Interrupt



Opcode	Clocks	Operation
CC	52	Interrupt 3
CD	51	Interrupt 'type'
CE	53 or 4	Interrupt4 if FLAGS.OF = 1, else NOP

IRET = Return from Interrupt



Opcode	Clocks	Operation
CF	24	Return from interrupt

Jcond = Jump on Condition

Operation

if condition is true then do;
 sign-extend displacement to 16 bits;
 IP ← IP + sign-extended displacement;
end if,

Format

Opcode	Disp
--------	------

Opcode	Clocks	Operation	cond
77	16 or 4	jump if above	JA
73	16 or 4	jump if above or equal	JAЕ
72	16 or 4	jump if below	JB
76	16 or 4	jump if below or equal	JBE
72	16 or 4	jump if carry set	JC
74	16 or 4	jump if equal	JE
7F	16 or 4	jump if greater	JG
7D	16 or 4	jump if greater or equal	JGE
7C	16 or 4	jump if less	JL
7E	16 or 4	jump if less or equal	JLE
76	16 or 4	jump if not above	JNA
72	16 or 4	jump if neither above nor equal	JNAЕ
73	16 or 4	jump if not below	JNB
77	16 or 4	jump if neither below nor equal	JNBE
73	16 or 4	jump if no carry	JNC
75	16 or 4	jump if not equal	JNE
7E	16 or 4	jump if not greater	JNG
7C	16 or 4	jump if neither greater nor equal	JNGE
7D	16 or 4	jump if not less	JNL
7F	16 or 4	jump if neither less nor equal	JNLE
71	16 or 4	jump if no overflow	JNO
7B	16 or 4	jump if no parity	JNP
79	16 or 4	jump if positive	JNS
75	16 or 4	jump if not zero	JNZ
70	16 or 4	jump if overflow	JO
7A	16 or 4	jump if parity	JP
7A	16 or 4	jump if parity even	JPE
7B	16 or 4	jump if parity odd	JPO
78	16 or 4	jump if sign	JS
74	18 or 6	jump if zero	JZ
E3	18 or 6	jump if CX is zero (does not test flags)	JCXZ

JMP = Jump

Within segment or group, IP relative

Opcode	DispL	DispH
--------	-------	-------

Opcode	Clocks	Operation
E9	15	$IP \leftarrow IP + \text{Disp16}$
EB	15	$IP \leftarrow IP + \text{Disp8}$ (Disp8 sign-extended)

Within segment or group, Indirect

Opcode	mod 100 r/m				
--------	-------------	--	--	--	--

Opcode	Clocks	Operation
FF	11	$IP \leftarrow \text{Reg16}$
FF	$18 + EA$	$IP \leftarrow \text{Mem16}$
FF	$18 + EA$	$IP \leftarrow \text{Mem16}$

Inter-segment or group, Direct

Opcode	offset	offset	segbase	segbase
--------	--------	--------	---------	---------

Opcode	Clocks	Operation
EA	15	$CS \leftarrow \text{segbase}$ $IP \leftarrow \text{offset}$

Inter-segment or group, Indirect

Opcode	mod 101 r/m				
--------	-------------	--	--	--	--

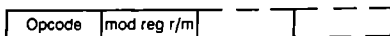
Opcode	Clocks	Operation
FF	$24 + EA$	$CS \leftarrow \text{segbase}$ $IP \leftarrow \text{offset}$

LAHF = Load AH from Flags

Opcode

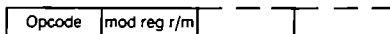
Opcode	Clocks	Operation
9F	4	copy low byte of flags word to AH

LDS/LES = Load Pointer to DS/ES and Register



Opcode	Clocks	Operation
C4	16 ÷ EA	dword pointer at EA goes to reg16 (1st word) and ES (2nd word)
C5	16 ÷ EA	dword pointer at EA goes to reg16 (1st word) and DS (2nd word)

LEA = Load Effective Address



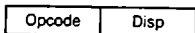
Opcode	Clocks	Operation
8D	2 ÷ EA	Reg16 ← EA

LOCK = Assert Bus Lock



Opcode	Clocks	Operation
F0	2	assert the bus lock next instruction

LOOPxx = Loop Control



Opcode	Clocks	Operation	xx =
E1	18 or 6	dec CX; loop if equal and CX not 0	LOOPE
E0	19 or 5	dec CX; loop if not equal and CX not 0	LOOPNE
E1	18 or 6	dec CX; loop if zero and CX not 0	LOOPZ
E0	19 or 5	dec CX; loop if not zero and CX not 0	LOOPNZ
E2	17 or 5	dec CX; loop if CX not 0	LOOP

MOV = Move Data

Memory/Reg to or from Reg

Opcode	mod reg r/m				
--------	-------------	--	--	--	--

	Opcode	Clocks	Operation
Byte	88	9 + EA	Mem8 $\leftrightarrow$ Reg8
	88	2	Reg8 $\leftrightarrow$ Reg8
	8A	8 + EA	Reg8 $\leftrightarrow$ Mem8
Word	89	9 + EA	Mem16 $\leftrightarrow$ Reg16
	89	2	Reg16 $\leftrightarrow$ Reg16
	8B	8 + EA	Reg16 $\leftrightarrow$ Mem16

Direct-Addressed Memory to or from AX/AL

Opcode	AddrL	AddrH
--------	-------	-------

	Opcode	Clocks	Operation
Byte	A0	10	AL $\leftrightarrow$ Mem8
	A2	10	Mem8 $\leftrightarrow$ AL
Word	A1	10	AX $\leftrightarrow$ Mem16
	A3	10	Mem16 $\leftrightarrow$ AX

Immed to Reg

Opcode	Data		
--------	------	--	--

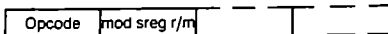
	Opcode	Clocks	Operation
Byte	B0 + reg	4	Reg 8 $\leftrightarrow$ Immed8
Word	B8 + reg	4	Reg16 $\leftrightarrow$ Immed16

Immed to Memory/Reg

Opcode	mod 000 r/m				Data		
--------	-------------	--	--	--	------	--	--

	Opcode	Clocks	Operation
	C6	4	Reg8 $\leftrightarrow$ Immed8
	C6	10 + EA	Mem8 $\leftrightarrow$ Immed8
	C7	4	Reg16 $\leftrightarrow$ Immed16
	C7	10 + EA	Mem16 $\leftrightarrow$ Immed16

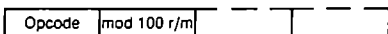
Memory/Reg to or from SReg



	Opcode	Clocks	Operation
Word	8C	9 ÷ EA	Mem16 ↔ SReg
	8C	2	Reg16 ↔ SReg
	8E	8 ÷ EA	SReg ↔ Mem16
	8E	2	SReg ↔ Reg16

MUL = Unsigned Multiplication

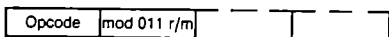
Memory/Reg with AL or AX



	Opcode	Clocks	Operation
Byte	F6	70-77	AX ↔ AL * Reg8
	F6	(76-83) ÷ EA	AX ↔ AL * Mem8
Word	F7	118-133	DX:AX ↔ AX * Reg16
	F7	(124-139) ÷ EA	DX:AX ↔ AX * Mem16

NEG = Negate an Integer

Memory/Reg



	Opcode	Clocks	Operation
	F6	3	Reg8 ↔ 00H - Reg 8
	F7	3	Reg16 ↔ 0000H - Reg16
	F6	16 ÷ EA	Mem8 ↔ 00H - Mem8
	F7	16 ÷ EA	Mem16 ↔ 0000H - Mem16

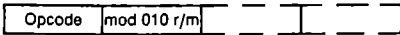
NOP = No Operation



Opcode	Clocks	Operation
90	3	no operation

NOT = Form One's Complement

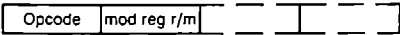
Memory/Reg



	Opcode	Clocks	Operation
Byte	F6	3	Reg8 $\leftarrow$ 0FFH - Reg8
	F6	16 + EA	Mem8 $\leftarrow$ 0FFH - Mem8
Word	F7	3	Reg16 $\leftarrow$ 0FFFFH - Reg16
	F7	16 + EA	Mem16 $\leftarrow$ 0FFFFH - Mem16

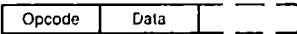
OR = Logical Inclusive OR

Memory/Reg with Reg



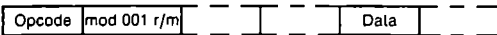
	Opcode	Clocks	Operation
Byte	0A	3	Reg8 $\leftarrow$ Reg8 OR Reg8
	0A	9 + EA	Reg8 $\leftarrow$ Reg8 OR Mem8
	0B	16 + EA	Mem8 $\leftarrow$ Mem8 OR Reg8
Word	0B	3	Reg16 $\leftarrow$ Reg16 OR Reg 16
	0B	9 + EA	Reg16 $\leftarrow$ Reg16 OR Mem16
	09	16 + EA	Mem16 $\leftarrow$ Mem16 OR Reg16

Immed to AX/AL



Opcode	Clocks	Operation
0C	4	AL $\leftarrow$ AL OR Immed8
0D	4	AX $\leftarrow$ AX OR Immed16

Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	80	4	Reg8 $\leftarrow$ Reg8 OR Immed8
	80	17 + EA	Mem8 $\leftarrow$ Mem8 OR Immed8
Word	81	4	Reg16 $\leftarrow$ Reg16 OR Immed16
	81	17 + EA	Mem16 $\leftarrow$ Mem16 OR Immed16

OUT = Output Byte, Word

Fixed port

Opcode	Port
--------	------

	Opcode	Clocks	Operation
Byte	E6	10	Port8 $\rightarrow$ AL
	E7	10	Port8 $\rightarrow$ AX

Variable port

Opcode

	Opcode	Clocks	Operation
Word	EE	8	Port16 (in DX) $\rightarrow$ AL
	EF	8	Port16 (in DX) $\rightarrow$ AX

POP = Pop a Word from the Stack

Word Memory

Opcode	mod 000 r/m				
--------	-------------	--	--	--	--

Opcode	Clocks	Operation
8F	17 + EA	Mem16 $\leftarrow$ (SP) + +

Word Register

Opcode + reg

Opcode	Clocks	Operation
58 + reg	8	Reg16 $\leftarrow$ (SP) + +

Segment Register

Opcode + SReg

Opcode	Clocks	Operation
07 + SReg	8	SReg $\leftarrow$ (SP) + +

POPF = Pop the TOS into the Flags

Opcode

Opcode	Clocks	Operation
9D	8	FLAGS $\leftarrow$ (SP)++

PUSH = Push a Word onto the Stack

Memory/Reg

Opcode	mod 110 r/m				
--------	-------------	--	--	--	--

Opcode	Clocks	Operation
FF	16 + EA	$-(SP) \leftarrow \text{Mem16}$

Word Register

Opcode + reg

Opcode	Clocks	Operation
50 + reg	11	$-(SP) \leftarrow \text{Reg16}$

Segment Register

Opcode + SReg

Opcode	Clocks	Operation
06 + SReg	10	$-(SP) \leftarrow \text{SReg}$

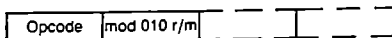
PUSHF = Push the Flags to the Stack

Opcode

Opcode	Clocks	Operation
9C	10	$-(SP) \leftarrow \text{FLAGS}$

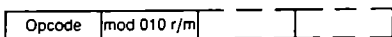
RCL = Rotate Left Through Carry

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg 8 by 1
	D0	15 ÷ EA	rotate Mem8 by 1
Word	D1	2	rotate Reg 16 by 1
	D1	15 ÷ EA	rotate Mem16 by 1

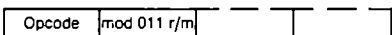
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8 ÷ 4/bit	rotate Reg8 by CL
	D2	20 ÷ EA ÷ 4/bit	rotate Mem8 by CL
Word	D3	8 ÷ 4/bit	rotate Reg16 by CL
	D3	20 ÷ EA ÷ 4/bit	rotate Mem16 by CL

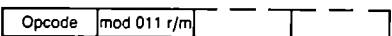
RCR = Rotate Right Through Carry

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15 ÷ EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15 ÷ EA	rotate Mem16 by 1

Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8 ÷ 4/bit	rotate Reg8 by CL
	D2	20 ÷ EA ÷ 4/bit	rotate Mem8 by CL
Word	D3	8 ÷ 4/bit	rotate Reg16 by CL
	D3	20 ÷ EA ÷ 4/bit	rotate Mem16 by CL

REP_x = Repeat Prefix

Opcode

Opcode	Clocks	Operation	REP _x
F3	2	repeat next instruction until CX := 0	REP
F3	2	repeat next instruction until CX = 0 or ZF := 1	REPE REPZ
F2	2	repeat next instruction until CX := 0 or ZF = 0	REPNE REPNZ

RET = Return from Subroutine

Opcode

Opcode	Clocks	Operation
C3	8	intra-segment return
CB	18	inter-segment return

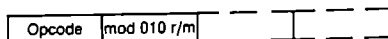
Return and add constant to SP

Opcode	DataL	DataH
--------	-------	-------

Opcode	Clocks	Operation
C2	12	intra-segment ret and add
CA	17	inter-segment ret and add

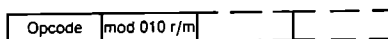
ROL = Rotate Left

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15 ÷ EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15 ÷ EA	rotate Mem16 by 1

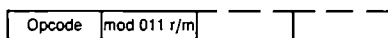
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8 ÷ 4/bit	rotate Reg8 by CL
	D2	20 ÷ Ea ÷ 4/bit	rotate Mem8 by CL
Word	D3	8 ÷ 4/bit	rotate Reg16 by CL
	D3	20 ÷ EA ÷ 4/bit	rotate Mem16 by CL

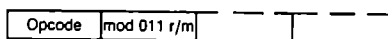
ROR = Rotate Right

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15 ÷ EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15 ÷ EA	rotate Mem16 by 1

Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8 ÷ 4/bit	rotate Reg8 by CL
	D2	20 ÷ EA ÷ 4/bit	rotate Mem8 by CL
	D3	8 ÷ 4/bit	rotate Reg16 by CL
	D3	20 ÷ EA ÷ 4/bit	rotate Mem16 by CL

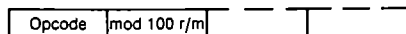
SAHF = Store AH in Flags

Opcode

Opcode	Clocks	Operation
9E	4	copy AH to low byte of flags word

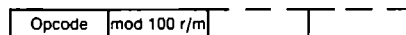
SAL/SHL = Arithmetic/Logical Left Shift

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15 + EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15 + EA	shift Mem16 by 1

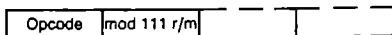
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	shift Reg8 by CL
	D2	20 + EA + 4/bit	shift Mem8 by CL
Word	D3	8 + 4/bit	shift Reg16 by CL
	D3	20 + EA + 4/bit	shift Mem16 by CL

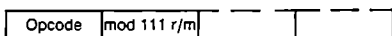
SAR = Arithmetic Right Shift

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	$15 \div EA$	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	$15 \div EA$	shift Mem16 by 1

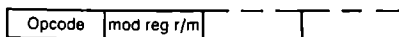
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	$8 \div 4/\text{bit}$	shift Reg8 by CL
	D2	$20 \div EA \div 4/\text{bit}$	shift Mem8 by CL
Word	D3	$8 \div 4/\text{bit}$	shift Reg16 by CL
	D3	$20 \div EA \div 4/\text{bit}$	shift Mem16 by CL

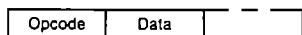
SBB = Integer Subtraction with Borrow

Memory/Reg with Reg



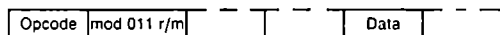
	Opcode	Clocks	Operation
Byte	1A	3	Reg8 $\leftarrow$ Reg8 - Reg8 - CF
	1A	9 + EA	Reg8 $\leftarrow$ Reg8 - Mem8 - CF
	1B	16 + EA	Mem8 $\leftarrow$ Mem8 - Reg8 - CF
Word	1B	3	Reg16 $\leftarrow$ Reg16 - Reg16 - CF
	1B	9 + EA	Reg16 $\leftarrow$ Reg16 - Mem16 - CF
	19	16 + EA	Mem16 $\leftarrow$ Mem16 - Reg16 - CF

Immed from AX/AL



Opcode	Clocks	Operation
1C	4	AL $\leftarrow$ AL - Immed8 - CF
1D	4	AX $\leftarrow$ AX - Immed16 - CF

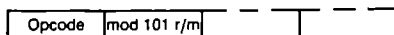
Immed from Memory/Reg



Opcode	Clocks	Operation
80	4	Reg8 $\leftarrow$ Reg8 - Immed8 - CF
80	17 + EA	Mem8 $\leftarrow$ Mem8 - Immed8 - CF
81	4	Reg16 $\leftarrow$ Reg16 - Immed16 - CF
81	17 + EA	Mem16 $\leftarrow$ Mem16 - Immed16 - CF
83	4	Reg16 $\leftarrow$ Reg16 - Immed8 - CF
83	17 + EA	Mem16 $\leftarrow$ Mem16 - Immed8 - CF (Immed8 is sign-extended before subtract)

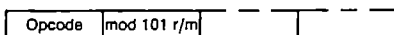
SHR = Logical Right Shift

Memory or Reg by 1



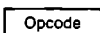
	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15 ÷ EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15 ÷ EA	shift Mem16 by 1

Memory or Reg by count in CL



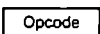
	Opcode	Clocks	Operation
Byte	D2	8 ÷ 4/bit	shift Reg8 by CL
	D2	20 ÷ Ea + 4/bit	shift Mem8 by CL
Word	D3	8 ÷ 4/bit	shift Reg16 by CL
	D3	20 ÷ EA + 4/bit	shift Mem16 by CL

STC = Set Carry Flag



Opcode	Clocks	Operation
F9	2	set the carry flag

STD = Set Direction Flags



Opcode	Clocks	Operation
FD	2	set direction flag

STI = Set Interrupt Enable Flag

Opcode

Opcode	Clocks	Operation
FB	2	set interrupt flag

String = String Operations

Opcode

Opcode	Clocks	Operation	String =
A6	22	flags $\leftrightarrow$ (SI) - (DI)	CMPS
A7	22	flags $\leftrightarrow$ (SI) - (DI)	CMPS
A4	18	(DI) $\leftrightarrow$ (SI)	MOVS
A5	18	(DI) $\leftrightarrow$ (SI)	MOVS
AE	15	flags $\leftrightarrow$ (DI) - AL	SCAS
AF	15	flags $\leftrightarrow$ (DI) - AX	SCAS
AC	12	AL $\leftrightarrow$ (SI)	LDS
AD	12	AX $\leftrightarrow$ (SI)	LDS
AA	11	(DI) $\leftrightarrow$ AL	STOS
AB	11	(DI) $\leftrightarrow$ AX	STOS

SUB = Integer Subtraction

Memory/Reg with Reg

Opcode	mod reg r/m		
--------	-------------	--	--

	Opcode	Clocks	Operation
Byte	2A	3	Reg8 Reg8 - Reg8
	2A	9 + EA	Reg8 Reg8 - Mem8
	2B	16 + EA	Mem8 Mem8 - Reg8
Word	2B	3	Reg16 Reg16 - Reg16
	2B	9 + EA	Reg16 Reg16 - Mem16
	29	16 + EA	Mem16 Mem16 - Reg16

Immed to AX/AL

Opcode	Data		
--------	------	--	--

	Opcode	Clocks	Operation
Byte	2C	4	AL $\leftarrow$ AL - Immed8
Word	2D	4	AX $\leftarrow$ AX - Immed16

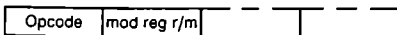
Immed to Memory/Reg

Opcode	mod 101 r/m							Data		
--------	-------------	--	--	--	--	--	--	------	--	--

	Opcode	Clocks	Operation
Byte	80	4	Reg8 $\leftarrow$ Reg8 - Immed8
	80	17 + EA	Mem8 $\leftarrow$ Mem8 - Immed8
Word	81	4	Reg16 $\leftarrow$ Reg16 - Immed16
	81	17 + EA	Mem16 $\leftarrow$ Mem16 - Immed16
	83	4	Reg16 $\leftarrow$ Reg16 - Immed8
	83	17 + EA	Mem16 $\leftarrow$ Mem16 - Immed8

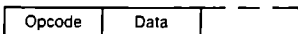
TEST = Logical Compare

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	84	3	flags $\leftarrow$ Reg8 AND Reg8
	84	9 + EA	flags $\leftarrow$ Reg8 AND Mem8
Word	85	3	flags $\leftarrow$ Reg16 AND Reg16
	85	9 + EA	flags $\leftarrow$ Reg16 AND Mem16

Immed to AX/AL



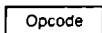
	Opcode	Clocks	Operation
Byte	A8	4	flags $\leftarrow$ AL AND Immed8
Word	A9	4	flags $\leftarrow$ AX AND Immed16

Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	F6	5	flags $\leftarrow$ Reg8 AND Immed8
	F6	11 + EA	flags $\leftarrow$ Mem8 AND Immed8
Word	F7	5	flags $\leftarrow$ Reg16 AND Immed16
	F7	11 + EA	flags $\leftarrow$ Mem16 AND Immed16

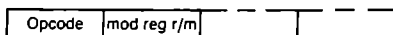
WAIT = Wait While TEST Pin Not Asserted



Opcode	Clocks	Operation
9B	3 + 5n	none

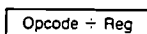
XCHG = Exchange Memory/Register with Register

Memory/Reg with Reg



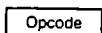
	Opcode	Clocks	Operation
Byte	86	4	Reg8 ↔ Reg8
	86	17 ÷ EA	Mem8 ↔ Mem8
Word	87	4	Reg16 ↔ Reg16
	87	17 ÷ EA	Mem16 ↔ Mem16

Word Register with AX



Opcode	Clocks	Operation
90 ÷ Reg	3	AX ↔ Reg16

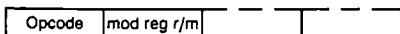
XLAT XLATB = Table Look-up Translation



Opcode	Clocks	Operation
D7	11	replace AL with table entry

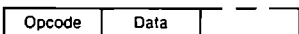
XOR = Logical Exclusive OR

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	32	3	Reg8 $\leftarrow$ Reg8 XOR Reg8
	32	9 + EA	Reg8 $\leftarrow$ Reg8 XOR Mem8
	30	16 + EA	Mem8 $\leftarrow$ Mem8 XOR Reg8
Word	33	3	Reg16 $\leftarrow$ Reg16 XOR Reg16
	33	9 + EA	Reg16 $\leftarrow$ Reg16 XOR Mem16
	31	16 + EA	Mem16 $\leftarrow$ Mem16 XOR Reg16

Immed to AX/AL



Opcode	Clocks	Operation
34	4	AL $\leftarrow$ AL XOR Immed8
35	4	AX $\leftarrow$ AX XOR Immed16

Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	80	4	Reg8 $\leftarrow$ Reg8 XOR Immed8
	80	17 + EA	Mem8 $\leftarrow$ Mem8 XOR Immed8
Word	81	4	Reg16 $\leftarrow$ Reg16 XOR Immed16
	81	17 + EA	Mem16 $\leftarrow$ Mem16 XOR Immed16

186 INSTRUCTIONS

Notes for iAPX 186 Instructions

These instructions can be used only if the MOD186 control is specified. When MOD186 is specified, clocks for all instructions are as stated under "Clocks for MOD186 Operation."

BOUND = Check Array Against Bounds

Opcode	ModRM				
--------	-------	--	--	--	--

Opcode Operation

62 if Reg16 < Mem16 at EA, or
Reg16 > Mem16 at EA + 2 then
INTERRUPT 5

ENTER = High Level Procedure Entry

Opcode	DataL	DataH	Level
--------	-------	-------	-------

Opcode Operation

C8 build new stack frame

IMUL = Signed Multiplication

Mem/Reg * Immediate to Reg

Opcode	ModRM					Data		
--------	-------	--	--	--	--	------	--	--

Opcode Operation

6B Reg 16 $\leftarrow$ Reg 16 * Immed 8
6B Reg 16 $\leftarrow$ Reg 16 * Immed 8
6B Reg 16 $\leftarrow$ Mem 16 * Immed 8
69 Reg 16 $\leftarrow$ Reg 16 * Immed 16
69 Reg 16 $\leftarrow$ Reg 16 * Immed 16
69 Reg 16 $\leftarrow$ Mem 16 * Immed 16

LEAVE $\approx$ High Level Procedure Exit

Opcode

Opcode Operation

C9 release current stack frame
and return to prior frame.

POPA = Pop All Registers

Opcode

Opcode Operation

61 restore registers from
stack

PUSH = Push a Word onto the Stack

Word Immediate

Opcode	Data			
--------	------	--	--	--

Opcode Operation

6A $-(SP) \leftarrow \text{Immed8}$
(sign extended)
68 $-(SP) \leftarrow \text{Immed16}$

PUSHA = Push All Registers

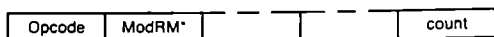
Opcode

Opcode Operation

60 save registers on the stack

RCL = Rotate Left Through Carry

Mem or Reg by Immed8



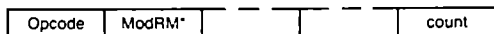
*—(Reg field = 011)

Opcode Operation

- C0 rotate Reg8 by Immed8
- C0 rotate Mem8 by Immed8
- C1 rotate Reg16 by Immed8
- C1 rotate Mem16 by Immed8

RCR = Rotate Right Through Carry

Mem or Reg by Immed8



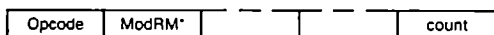
*—(Reg field = 011)

Opcode Operation

- C0 rotate Reg8 by Immed8
- C0 rotate Mem8 by Immed8
- C1 rotate Reg16 by Immed8
- C1 rotate Mem16 by Immed8

ROL = Rotate Left

Mem or Reg by Immed8



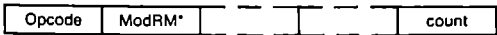
*—(Reg field = 000)

Opcode Operation

- C0 rotate Reg8 by Immed8
- C0 rotate Mem8 by Immed8
- C1 rotate Reg16 by Immed8
- C1 rotate Mem16 by Immed8

ROR = Rotate Right

Mem or Reg by Immed8



*—(Reg field = 001)

Opcode Operation

- C0 rotate Reg8 by Immed8
- C0 rotate Mem8 by Immed8
- C1 rotate Reg16 by Immed8
- C1 rotate Mem16 by Immed8

SAL/SHL = Arithmetic/Logical Left Shift

Mem or Reg by immediate count



*—(Reg field = 100)

Opcode Operation

- C0 rotate Reg8 by Immed8
- C0 rotate Mem8 by Immed8
- C1 rotate Reg16 by Immed8
- C1 rotate Mem16 by Immed8

SAR = Arithmetic Right Shift

Mem or Reg by Immed8



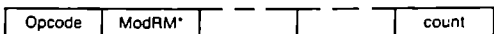
*—(Reg field = 111)

Opcode Operation

- C0 rotate Reg8 by Immed8
- C0 rotate Mem8 by Immed8
- C1 rotate Reg16 by Immed8
- C1 rotate Mem16 by Immed8

SHR = Logical Right Shift

Mem or Reg by Immed8

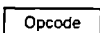


*—(Reg field = 101)

Opcode Operation

- C0 rotate Reg8 by Immed8
- C0 rotate Mem8 by Immed8
- C1 rotate Reg16 by Immed8
- C1 rotate Mem16 by Immed8

String = String Operations (INS/OUTS)



Opcode	Clocks	Operation
6E	INS	(DI) $\leftrightarrow$ port(DX)
6F	INS	(DI) $\leftrightarrow$ port(DX·DX + 1)
6C	OUTS	port(DX) $\leftrightarrow$ (SI)
6D	OUTS	port(DX·DX + 1) $\leftrightarrow$ (SI)

8087 INSTRUCTIONS

Notes for 8087 Instructions

The individual instruction descriptions are shown by a format box such as the following:

WAIT	op1	m/op/r/m	addr1	addr2
------	-----	----------	-------	-------

These are the byte-wise representations of the object code generated by the assembler and are interpreted as follows:

- WAIT is an 8086 wait instruction, NOP or emulator instruction.
- op1 is the opcode, possibly taking two bytes.
- m/op/r/m byte (middle 3-bits is part of the opcode).
- addr1 and addr2 are offsets of either 8 or 16 bits.

For integer functions, m = 0 for short-integer memory operand; 1 for word-integer memory operand.

For real functions, m = 0 for short-real memory operand; 1 for longreal memory operand.

i = stack element index.

If mod = 00 then DISP = 0, disp-lo and disp-hi are absent

If mod = 01 then DISP = disp-lo sign-extended to 16 bits, disp-hi is absent.

If mod = 10 then DISP = disp-hi, disp-lo

If mod = 11 then r/m is treated as an ST(i) field.

If r/m = 000 then EA = (BX) + (SI) + DISP

If r/m = 001 then EA = (BX) + (DI) + DISP

If r/m = 010 then EA = (BP) + (SI) + DISP

If r/m = 011 then EA = (BP) + (DI) + DISP

If r/m = 100 then EA = (SI) + DISP

If r/m = 101 then EA = (DI) + DISP

If r/m = 110 then EA = (BP) + DISP*

If r/m = 111 then EA = (BX) + DISP

*Except if mod = 000 and r/m = 110 then EA = disp-hi; disp-lo.

ST(0) = Current stack top

ST(i) = ith register below stack top

d = Destination

0 — Destination is ST(0)

1 — Destination is ST(i)

P = Pop

0 — No pop

1 — Pop ST(0)

R = Reverse

0 — Destination (op) source

1 — Source (op) destination

For FSQRT: $-0 \leq ST(0) \leq +\infty$
 For FSCALE: $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
 For F2XM1: $0 \leq ST(0) \leq 2^{-1}$
 For FYL2X: $0 \leq ST(0) < \infty$
 $-\infty < ST(1) < +\infty$
 For FYL2XP1: $0 < |ST(0)| < (2 - \sqrt{2})/2$
 $-\infty \leq ST(1) < \infty$
 For FPTAN: $0 \leq ST(0) < \pi/4$
 For FPATAN: $0 \leq ST(0) < ST(1) < +\infty$

F2XMI = Compute $2^x - 1$

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 F0	CD 19 F0	500 310-630	$ST \leftarrow 2^{ST(1)} - 1$

FABS = Absolute Value

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 E1	CD 19 E1	14 10-17	$ST \leftarrow ST $

FADD = Add Real

Stack top + Stack element

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 C0 + i	CD 18 C0 + i	85 70-100	ST $\rightarrow$ ST + ST(i)
9B DC C0 + i	CD 1C C0 + i	85 70-100	ST(i) $\rightarrow$ ST + ST(i)

Stack top + memory operand

WAIT	op1	mod 000 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 m0rm	CD 18 m0rm	105 + EA (90-120) + EA	ST $\rightarrow$ ST + mem-op (short-real)
9B DC m0rm	CD 1C m0rm	110 + EA (95-125) + EA	ST $\rightarrow$ ST + mem-op (long-real)

FADDP = Add Real and Pop

Stack top + Stack Element

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DE C1	CD 1E C1	90 75-105	ST(1) $\rightarrow$ ST + ST(1) pop stack
9B DE C0 + i	CD 1E C0 + i	90 75-105	ST(i) $\rightarrow$ ST + ST(i) pop stack

FBLD = Packed Decimal (BCD) Load

WAIT	op1	mod 100 r/m	addr1	addr2
------	-----	-------------	-------	-------

Execution
Clocks

8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DF m4rm	CD 1F m4rm	$300 \div EA$ $(290-310) \div EA$	push stack $ST \leftarrow mem-op$

FBSTP = Packed Decimal (BCD) Store and Pop

WAIT	op1	mod 110 r/m	addr1	addr2
------	-----	-------------	-------	-------

Execution
Clocks

8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DF m6rm	CD 1F m6rm	$530 \div EA$ $(520-540) \div EA$	$mem-op \leftarrow ST$ pop stack

FCBS = Change Sign

WAIT	op1	op2
------	-----	-----

Execution
Clocks

8087 Encoding	Emulator Encoding	Typical Range	Operation
9B D9 E0	CD 19 E0	15 10-17	$ST \leftarrow -ST$

FCLEX = Clear Exceptions

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DB E2	CD 1B E2	5 2-8	clear 8087 exceptions
90 DB E2	CD 1B E2	5 2-8	clear 8087 exceptions (no wait)

FCOM = Compare Real

Compare Stack top and Stack element

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 D1	CD 18 D1	45 40-50	ST — ST(1)
9B D8 D0+i	CD 18 D0+i	45 40-50	ST — ST(i)

Compare Stack top and memory operands

WAIT	op1	mod 010 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 m2rm	CD 18 m2rm	65 + EA (60-70) + EA	ST — memop (short-real)
9B DC m2rm	CD 1C m2rm	70 + EA (65-75) + EA	ST — memop (long-real)

FCOMP = Compare Real and Pop

Compare Stack top and Stack element and pop

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 D9	CD 18 D9	47 42-52	ST — ST(1) pop stack
9B D8 D8 + i	CD 18 D8 + i	47 42-52	ST — ST(i) pop stack

Compare Stack top and memory operand and pop

WAIT	op1	mod 011 r/m	addr1		addr2
------	-----	-------------	-------	--	-------

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m3rm	CD 18 m3rm	68 ÷ EA (63-73) + EA	ST — mem-op pop stack (short-real)
9B DC m3rm	CD 1C m3rm	72 ÷ EA (67-77) + EA	ST — mem-op pop stack (long-real)

FCOMPP = Compare Real and Pop Twice

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE D9	CD 1E D9	50 45-55	ST — ST(1) pop stack pop stack

FDECSTP = Decrement Stack Pointer

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 F6	CD 19 F6	9 6-12	stack pointer ← stack pointer - 1

FDISI FNDISI = Disable Interrupts

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DB E1	CD 1B E1	5 2-8	Set 8087 interrupt mask
90 DB E1	CD 1B E1	5 2-8	Set 8087 interrupt mask (no wait)

FDIV = Divide Real

Stack top and Stack element

WAIT	op1	op2 ÷ i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 F0 ÷ i	CD 18 F0 ÷ i	198 193-203	ST → ST/ST(i)
9B DC F8 ÷ i	CD 1C F8 ÷ i	198 193-203	ST(i) → ST(i)/ST

Stack top and memory operand

WAIT	op1	mod 110 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 m6rm	CD 18 m6rm	220 + EA (215-225) + EA	ST → ST/mem-op (short-real)
9B DC m6rm	CD 1C m6rm	225 + EA (220-230) + EA	ST → ST/mem-op (long-real)

FDIVP = Divide Real and Pop

WAIT	op1	op2 ÷ i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DE F9	CD 1E F9	202 197-207	ST(1) → ST(1)/ST pop stack
9B DE F8 ÷ i	CD 1E F8 ÷ i	202 197-207	ST(i) → ST(i)/ST pop stack

FDIVR = Divide Real Reversed

Stack top and Stack element

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 F8+i	CD 18 F8+i	199 194-204	ST $\leftarrow$ ST(i)/ST
9B DC F0+i	CD 1C F0+i	199 194-204	ST(i) $\leftarrow$ ST/ST(i)

Stack top and memory operand

WAIT	op1	mod 111 r/m	addr1		addr2
------	-----	-------------	-------	--	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 m7rm	CD 18 m7rm	221 + EA (216-226) + EA	ST $\leftarrow$ mem-op/ST (short-real)
9B DC m7rm	CD 1C m7rm	226 + EA (221-231) + EA	ST $\leftarrow$ mem-op/ST (long-real)

FDIVRP = Divide Real Reversed and Pop

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DE F1	CD 1E F1	203 198-208	ST(1) $\leftarrow$ ST/ST(1) pop stack
9B DE F0+i	CD 1E F0+i	203 198-208	ST(i) $\leftarrow$ ST/ST(i)

FENI = Enable Interrupts

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB E0	CD 1B E0	5 2-8	clear 8087 interrupt mask
90 DB E0	CD 1B E0	5 2-8	clear 8087 interrupt mask (no wait)

FFREE = Free Register

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD C0 + i	CD 1D C0 + i	11 9-16	TAG(i) masked empty

FIADD = Integer Add

WAIT	op1	mod 000 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m0rm	CD 1A m0rm	125 + EA (108-143) + EA	ST $\rightarrow$ ST + mem-op (short integer)
9B DE m0rm	CD 1E m0rm	120 + EA (102-137) + EA	ST $\rightarrow$ ST + mem-op (word integer)

FICOM = Integer Compare

WAIT	op1	mod 010 r/m	addr1		addr2
------	-----	-------------	-------	--	-------

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m2rm	CD 1A m2rm	$85 \div \text{EA}$ (78-91) + EA	ST ← mem-op (short integer)
19B DE m2rm	CD 1E m2rm	$80 \div \text{EA}$ (72-86) + EA	ST ← mem-op (word integer)

FICOMP = Integer Compare and Pop

WAIT	op1	mod 011 r/m	addr1		addr2
------	-----	-------------	-------	--	-------

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m3rm	CD 1A m3rm	$87 \div \text{EA}$ (80-93) + EA	ST ← mem-op pop stack (short integer)
9B DE m3rm	CD 1E m3rm	$82 \div \text{EA}$ (74-88) + EA	ST ← mem-op pop stack (word integer)

FIDIV = Integer Divide

WAIT	op1	mod 110 r/m	addr1		addr2
------	-----	-------------	-------	--	-------

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m6rm	CD 1A m6rm	$236 \div \text{EA}$ (230-243) + EA	ST ← ST/mem-op (short integer)
9B DE m6rm	CD 1E m6rm	$230 \div \text{EA}$ (224-238) + EA	ST ← ST/mem-op (word integer)

FIDIVR = Integer Divide Reversed

WAIT	op1	mod 111 r/m	addr1		addr2
------	-----	-------------	-------	--	-------

Execution Clocks

8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DA m7rm	CD 1A m7rm	$237 \div EA$ $(231-245) \div EA$	ST $\leftarrow$ mem-op/ST (short integer)
9B DE m7rm	CD 1E m7rm	$230 \div EA$ $(225-239) \div EA$	ST $\leftarrow$ mem-op/ST (word integer)

FILD = Integer Load

Word Integer or Short Integer

WAIT	op1	mod 000 r/m	addr1		addr2
------	-----	-------------	-------	--	-------

Execution Clocks

8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DB m0rm	CD 1B m0rm	$56 \div EA$ $(52-60) \div EA$	push stack ST $\leftarrow$ mem-op (short integer)
9B DF m0rm	CD 1F m0rm	$50 \div EA$ $(46-54) \div EA$	push stack ST $\leftarrow$ mem-op (word integer)

Long Integer

WAIT	op1	mod 101	addr1		addr2
------	-----	---------	-------	--	-------

Execution Clocks

8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DF m5rm	CD 1F m5rm	$64 \div EA$ $(60-68) \div EA$	push stack ST $\leftarrow$ mem-op (long integer)

FIMUL = Integer Multiply

WAIT	op1	mod001 r/m	addr1	addr2
------	-----	------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DA m1rm	CD 1A m1rm	136 ÷ EA (130-144) ÷ EA	ST ← ST * mem-op (short integer)
9B DE m1rm	CD 1E m1rm	130 ÷ EA (124-138) ÷ EA	ST ← ST * mem-op (word integer)

FINCSTP = Increment Stack Pointer

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 F7	CD 19 F7	9 6-12	stack pointer ← stack pointer + 1

FINIT = Initialize Processor

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DB E3	CD 1B E3	5 2-8	initialize 8087
90 DB E3	CD 1B E3	5 2-8	initialize 8087 (no wait)

FIST = Integer Store

WAIT	op1	mod 010 r/m	addr1	addr2
------	-----	-------------	-------	-------

		Execution Clocks	
8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DB m2rm	CD 1B m2rm	$88 + EA$ $(82-92) + EA$	mem-op $\leftrightarrow$ ST (short integer)
9B DF m2rm	CD 1F m2rm	$86 + EA$ $(80-90) + EA$	mem-op $\leftrightarrow$ ST (word integer)

FISTP = Integer Store and Pop

Short Integer or Word Integer

WAIT	op1	mod 011 r/m	addr1	addr2
------	-----	-------------	-------	-------

		Execution Clocks	
8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DB m3rm	CD 1B m3rm	$90 + EA$ $(84-94) + EA$	mem-op $\leftrightarrow$ ST pop stack (short integer)
9B DF m3rm	CD 1F m3rm	$88 + EA$ $(82-92) + EA$	mem-op $\leftrightarrow$ ST pop stack (word integer)

Long Integer

WAIT	op1	mod 111	addr1	addr2
------	-----	---------	-------	-------

		Execution Clocks	
8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DF m7rm	CD 1F m7rm	$100 + EA$ $(94-105) + EA$	mem-op $\leftrightarrow$ ST pop stack (long integer)

FISUB = Integer Subtract

WAIT	op1	mod 100 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DA m4rm	CD 1A m4rm	125 + EA (108-143) + EA	ST ← ST — mem-op (short integer)
9B DE m4rm	CD 1E m4rm	120 + EA (102-137) + EA	ST ← ST — mem-op (word integer)

FISUBR = Integer Subtract Reversed

WAIT	op1	mod 101 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DA m5rm	CD 1A m5rm	125 + EA (109-144) + EA	ST ← mem-op — ST (short integer)
9B DE m5rm	CD 1E m5rm	120 + EA (103-139) + EA	ST ← mem-op — ST (word integer)

FLD = Load Real

Stack element to Stack top

WAIT	op1	op2 $\leftarrow$ i
------	-----	--------------------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 C0 $\div$ i	CD 19 C0 $\div$ i	20 17-22	$T_i \leftarrow ST(i)$ push stack $ST \leftarrow T_i$

Memory operand to Stack top
Short Integer or Long Integer

WAIT	op1	mod 000 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 m0rm	CD 19 m0rm	43 + EA (38-56) + EA	push stack $ST \leftarrow \text{mem-op}$ (short integer)
9B DD m0rm	CD 1D m0rm	46 + EA (40-60) + EA	push stack $ST \leftarrow \text{mem-op}$ (long integer)

Temp Real

WAIT	op1	mod 101	addr1	addr2
------	-----	---------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DB m5rm	CD 1B m5rm	57 + EA (53-65) + EA	push stack $ST \leftarrow \text{mem-op}$ (temp real)

FLD1 = Load + 1.0

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E8	CD 19 E8	18 15-21	push stack ST $\leftarrow$ 1.0

FLDCW = Load Control Word

WAIT	op1	mod 101 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m5rm	CD 19 m5rm	10 + EA (7-14) + EA	processor control word $\leftarrow$ mem-op

FLDENV = Load Environment

WAIT	op1	mod 100 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m4rm	CD 19 m4rm	40 + EA (35-45) + EA	8087 environment $\leftarrow$ mem-op

FLDL2E = Load Log₂e

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 EA	CD 19 EA	18 15-21	push stack ST $\leftarrow$ log ₂ e

FLDL2T = Load $\log_2 10$

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 E9	CD 19 E9	19 16-22	push stack $ST \leftarrow \log_2 10$

FLDLG2 = Load $\log_{10} 2$

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 EC	CD 19 EC	21 18-24	push stack $ST \leftarrow \log_{10} 2$

FLDPI = Load π

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 EB	CD 19 EB	19 16-22	push stack $ST \leftarrow \pi$

FLDZ = Load + 0.0

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 EE	CD 19 EE	14 11-17	push stack $ST \leftarrow 0.0$

FMUL = Multiply Real

Stack top and Stack element

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 C8 + i	CD 18 C8 + i	138 130-145	ST ← ST * ST(i)
9B DC C8 + i	CD 1C C8 + i	138 130-145	ST(i) ← ST(i) * ST

Stack top and memory operand

WAIT	op1	mod 001 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 m1rm	CD 18 m1rm	118 + EA (110-125) + EA	ST ← ST * mem-op (short real)
9B DC m1rm	CD 1C m1rm	161 + EA (154-168) + EA	ST ← ST * mem-op (long real)

FMULP = Multiply Real and Pop

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DE C9 + i	CD 1E C9 + i	142 134-148	ST(i) ← ST(i) * ST pop stack

FNOP = No Operation

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 D0	CD 19 D0	13 10-16	ST $\leftarrow$ ST

FPATAN = Partial Arctangent

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 F3	CD 19 F3	650 250-800	$T_1 \leftarrow \arctan (ST(1)/ST)$ pop stack ST $\leftarrow T_1$

FPREM = Partial Remainder

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 F8	CD 19 F8	125 15-190	ST $\leftarrow$ REPEAT (ST — ST(1))

FPTAN = Partial Tangent

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 F2	CD 19 F2	450 30-540	$Y/X \leftarrow \tan (ST)$ ST $\leftarrow Y$ push stack ST $\leftarrow X$

FRNDINT = Round to Integer

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks		Operation
		Typical Range		
9B D9 FC	CD 19 FC	45	ST ← nearest integer (ST)	
		16-50		

FRSTOR = Restore Saved State

WAIT	op1	mod 100 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DD m4rm	CD 1D m4rm	202 + EA (197-207) + EA	8087 state $\leftrightarrow$ mem-op

FSAVE = Save State

WAIT	op1	mod 110 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks		Operation
		Typical Range		
9B DD m6rm	CD 1D m6rm	202 + EA (197-207) + EA	mem-op → 8087 state	
90 DD m6rm	CD 1D m6rm	202 + EA (197-207) + EA	mem-op → 8087 state (no wait)	

FSCALE = Scale

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 FD	CD 19 FD	35 32-38	$ST \leftarrow ST * 2^{op1}$

FSQRT = Square Root

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 FA	CD 19 FA	183 180-186	$ST \leftarrow \sqrt{ST}$

FST = Store Real

Stack top to Stack element

WAIT	op1	op2 ÷ i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DD D0 ÷ i	CD 1D D0 ÷ i	18 15-22	ST(i) ← ST

Stack top to memory operand

WAIT	op1	mod 010 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 m2rm	CD 19 m2rm	87 ÷ EA (84-90) ÷ EA	mem-op ← ST (short-real)
9B D0 m2rm	CD 1D m2rm	100 ÷ EA (96-104) ÷ EA	mem-op ← ST (long-real)

FSTCW = Store Control Word

WAIT	op1	mod 111 r/m	addr1	addr2
------	-----	-------------	-------	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 m7rm	CD 19 m7rm	15 ÷ EA (12-18) ÷ EA	mem-op ← processor control word
90 D9 m7rm	CD 19 m7rm	15 ÷ EA (12-18) ÷ EA	mem-op ← processor control word (no wait)

FSTENV FNSTENV = Store Environment

WAIT	op1	mod 110 r/m	addr1	addr2
------	-----	-------------	-------	-------

		Execution Clocks	
8087 Encoding	Emulator Encoding	Typical Range	Operation
9B D9 m6rm	CD 19 m6rm	45 + EA (40-50) + EA	mem-op ← 8087 environment
90 D9 r6rm	CD 19 m6rm	45 + EA (40-50) + EA	mem-op ← 8087 environment (no wait)

FSTP = Store Real and Pop

Stack top to Stack element

WAIT	op1	op2 + 1
------	-----	---------

		Execution Clocks	
8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DD D8+i	CD 1D D8+i	20 17-24	ST(i) ← ST pop stack

Stack top to memory operand

WAIT	op1	mod 011 r/m	addr1	addr2
------	-----	-------------	-------	-------

Long Real or Short Real

8087 Encoding	Emulator Encoding	Execution Clocks		Operation
		Typical Range		
9B D9 m3rm	CD 19 m3rm	89 + EA (86-92) + EA		mem-op ← ST pop stack (short-real)
9B DB m3rm	CD 1B m3rm	102 + EA (98-106) + EA		mem-op ← ST pop stack (long-real)

Temp Real

WAIT	op1	mod 111 r/m	disp-lo	disp-hi
------	-----	-------------	---------	---------

8087 Encoding	Emulator Encoding	Execution Clocks		Operation
		Typical Range		
9B DD m7rm	CD 1D m7rm	55 + EA	mem-op → ST	pop stack (temp-real)
		(52-58) + EA		

FSTSW = Store Status Word

WAIT	op1	mod 111 r/m	addr1	addr2
------	-----	-------------	-------	-------

		Execution Clocks	
8087 Encoding	Emulator Encoding	Typical Range	Operation
9B DD m7rm	CD 1D m7rm	15 + EA (12-18) + EA	mem-op $\leftarrow$ 8087 status word
90 DD m7rm	CD 1D m7rm	15 + EA (12-18) + EA	mem-op $\leftarrow$ 8087 status word (no wait)

FSUB = Subtract Real

Stack top and Stack element

WAIT	op1	op2 + i
------	-----	---------

		Execution Clocks	
8087 Encoding	Emulator Encoding	Typical Range	Operation
9B D8 E0 + i	CD 18 E0 + i	85 70-100	ST $\leftarrow$ ST — ST(i)
9B DC E8 + i	CD 1C E8 + i	85 70-100	ST(i) $\leftarrow$ ST(i) — ST

Stack top and memory operand

WAIT	op1	mod 100 r/m	addr1	addr2
------	-----	-------------	-------	-------

		Execution Clocks	
8087 Encoding	Emulator Encoding	Typical Range	Operation
9B D8 m4rm	CD 18 m4rm	105 + EA (90-120) + EA	ST $\leftarrow$ ST — mem-op (short-real)
9B DC m4rm	CD 1C m4rm	110 + EA (95-125) + EA	ST $\leftarrow$ ST — mem-op (long-real)

FSUBP = Subtract Real and Pop

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DE E9	CD 1E E9	90 75-105	$ST(1) \leftarrow ST(1) - ST$ pop stack
9B DE E8 + i	CD 1E E8 + i	90 75-105	$ST(i) \leftarrow ST(i) - ST$ pop stack

FSUBR = Subtract Real Reversed

Stack top and Stack element

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 E8 + i	CD D8 E8 + i	87 70-100	$ST \leftarrow ST(i) - ST$
9B DC E0 + i	CD 1C E0 + i	87 70-100	$ST(i) \leftarrow ST - ST(i)$

Stack top and memory operand

WAIT	op1	mod 101 r/m	addr1		addr2
------	-----	-------------	-------	--	-------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D8 m5rm	CD 18 m5rm	$105 + EA$ $(90-120) + EA$	$ST \leftarrow mem-op - ST$ (short-real)
9B DC m5rm	CD 1C m5rm	$110 + EA$ $(95-125) + EA$	$ST \leftarrow mem-op - ST$ (long-real)

FSUBRP = Subtract Real Reversed and Pop

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B DE E1	CD 1E E1	90 75-105	ST(1) ← ST — ST(1) pop stack
9B DE E0 + i	CD 1E E0 + i	90 75-105	ST(i) ← ST — ST(i) pop stack

FTST = Test Stack Top Against + 0.0

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 E4	CD 19 E4	42 38-48	ST ← ST — 0.0

FWAIT = (CPU) Wait While 8087 Is Busy

WAIT

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B	90	3 + 5n 3 + 5n	8086 wait instruction

FXAM = Examine Stack Top

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 E5	CD 19 E5	17 12-23	set condition code

FXCH = Exchange Registers

WAIT	op1	op2 + i
------	-----	---------

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 C8	CD 19 C8	12 10-15	$T_1 \leftrightarrow ST(1)$ $ST(1) \leftrightarrow ST$ $ST \leftrightarrow T_1$
9B D9 C8 + i	CD 19 C8 + i	12 10-15	$T_1 \leftrightarrow ST(i)$ $ST(i) \leftrightarrow ST$ $ST \leftrightarrow T_1$

FXTRACT = Extract Exponent and Significand

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 F4	CD 19 F4	50 27-55	$T_1 \leftarrow \text{exponent (ST)}$ $T_2 \leftarrow \text{significand (ST)}$ $ST \leftarrow T_1$ push stack $ST \leftarrow T_2$

FYL2X = Compute $Y \cdot \log_2 X$

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 F1	CD 19 F1	950 900-1100	$T_1 \leftarrow ST(1) \cdot \log_2 (ST)$ pop stack $ST \leftarrow T_1$

FYL2XP1 = Compute $Y \cdot \log_2 (X + 1)$

WAIT	op1	op2
------	-----	-----

8087 Encoding	Emulator Encoding	Execution Clocks	Operation
		Typical Range	
9B D9 F9	CD 19 F9	850 700-1000	$T_1 \leftarrow ST + 1$ $T_2 \leftarrow ST(1) \cdot \log_2 T_1$ pop stack $ST \leftarrow T_2$

Assembler Controls Summary

Default control shown in italics

PRIMARY CONTROLS

Control	Effect
DATE(d)	System Date
DEBUG/ <i>NODEBUG</i> DA DB/ <i>NODB</i>	DEBUG puts local symbols information into object file for debugging. <i>NODEBUG</i> suppresses loading of local symbols information.
ERRORPRINT/ <i>NOERRORPRINT</i> EP/ <i>NOEP</i>	ERRORPRINT creates a file containing a listing of source line errors. <i>NOERRORPRINT</i> suppresses creation of that file.
<i>MACRO</i> / <i>NOMACRO</i> <i>MR</i> / <i>NOMR</i>	MACRO specifies that macro processor language will be recognized in source files. <i>NOMACRO</i> specifies nonrecognition of macros. They are scanned as is normal assembly language.
MOD186/ <i>8086 mode</i> M1	MOD186 specifies that the iAPX 186 instruction set be recognized. The default is 8086 instructions only.
<i>OBJECT</i> / <i>NOOBJECT</i> <i>OJ</i> / <i>NOOJ</i>	OBJECT specifies the creation of an object module in the file specified. <i>NOOBJECT</i> specifies that an object module is not to be created.
PAGELength(n) PL(n)	Specifies number (n) of printed lines per page in print file. Minimum pagelength is 20. Default is 60 lines per page.
PAGEWIDTH (n) PW (n)	Specifies the number (n) of characters, or columns, per line in the print and the errorprint files. Minimum is 60, maximum is 255. Default is 120.
<i>PAGING</i> / <i>NOPAGING</i> <i>PI</i> / <i>NOPI</i>	PAGING specifies that print file is to be formatted into pages with header at top of each page. <i>NOPAGING</i> specifies no formatting into pages.

PRINT/NOPRINT
PR/NOPR

PRINT specifies that a source listing will be created during assembly. If no filename is specified, the source listing is written to the source file with the extension .LST appended. NOPRINT specifies that no source listing will be created.

SYMBOLS/NOSYMBOLS
SB/NOSB

SYMBOLS specifies that a symbol listing table will be appended to the source listing in print file. NOSYMBOLS suppresses symbol table listing.

TYPE/NOTYPE
TY/NOTY

TYPE specifies that type information be put into the object module. NOTYPE specifies that no type information be put into the object module

WORKFILES
WF

WORKFILES specifies the devices or directories used for storage of assembler-created temporary workfiles.

XREF/NOXREF
XR/NOXR

XREF specifies that a symbol table, including line numbers, be appended to the source listing in print file. NOXREF specifies that no cross-reference line numbers are to be included.

GENERAL CONTROLS

EJECT
EJ

Next line of source listing to be placed on new page

GEN/GENONLY/NOGEN
GE/GO/NOGE

Specify mode of listing assembler source text, macro calls and macro text in print file. GEN produces a listing that includes all source text, macro calls and expansion of each macro. GENONLY produces a listing that includes only source file non-macro text, and final result text for each macro called. NOGEN produces a listing that includes only the source file text.

INCLUDE
IC

Causes subsequent source lines to be input from specified file.

LIST/NOLIST
LI/NOLI

LIST specifies that listing of source program in print file is to resume with next source line read. NOLIST specifies that listing of source program in print file, beginning with next source line, is to be suppressed.

SAVE/RESTORE
SA/RS

SAVE specifies that current setting of general controls be saved on a stack. RESTORE specifies that general controls be set to values stored on stack.

TITLE
TT

Specifies the character string to appear on page header. Default title is module name specified in assembler NAME directive.

ASM86 Invocation

Under Series-III

```
- [:dev:] RUN [:dev:] ASM86 
 [:dev:] filename [ controls ] < cr >
```

You may also use RUN alone, and then enter ASM86 without the RUN CUSP as often as you wish:

```
- [:dev:] RUN < cr >
SERIES-III RUN 8086, Vx.y
>
```

At the right angle-bracket prompt, you may enter:

```
> [:dev:] ASM86 [:dev:] yourprogram [ controls ] < cr >
```

Linking ASM86 programs to 8087.LIB, E8087, E8087.LIB:

```
> [:dev:] LINK86 [:dev:] prog.obj, [:dev:] 
 E8087, [:dev:] E8087.LIB [ TO 
 [:dev:] prog.LNK ]
```

This command links your program, *prog.obj*, to the 8087 emulator, *E8087*, and its associated interface library, *E8087.LIB*. Use this command if your system does not contain an 8087 Numeric Data Processor.

If you have an 8087 NDP, use this command:

```
> [:dev.] LINK 86 [:dev.] prog.obj, 
     [:dev.] E8087.LIB [TO [:dev.]
    prog.LNK]
```

This links the 8087 interface to your program.

Under Series IV

```
{ logicalname[ ( ' / ' directoryname ) ' / ' ]
  or
  { ' / ' directoryname } ' / ' } ASM86 
 program name[ { controls } . . . ]
```

Under iRMX™ 86

- [*directory*] ASM86 *sourcepath* [*controls*]

Assembler Directives

Symbol Definition:

EQU
LABEL
PURGE

Memory Reservation and Data Definition:

DB
DW
DD
DQ
DT
RECORD

Location Counter and Segmentation Control:

SEGMENT/ENDS
ORG
GROUP
ASSUME
PROC/ENDP
CODEMACRO/ENDM

Program Linkage:

NAME
PUBLIC
EXTRN
END

Processor Reset Register Initialization

Flags = 0000H (to disable interrupts
and single-stepping)

CS = FFFFH
IP = 0000H (to begin execution at FFFF0H)

DS = 0000H
SS = 0000H
ES = 0000H

No other registers are acted upon during reset.

MCS[®]-86 Reserved Locations

Reserved Memory Locations

Intel Corporation reserves the use of memory location FFFF0H through FFFFFH (with the exception of FFFF0H - FFFF5H for JMP instr.) for Intel hardware and software products. If you use these locations for some other purpose, you may preclude compatibility of your system with certain of these products.

Reserved Input/Output Locations

Intel Corporation reserves the use of input/output locations F8H through FFH for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

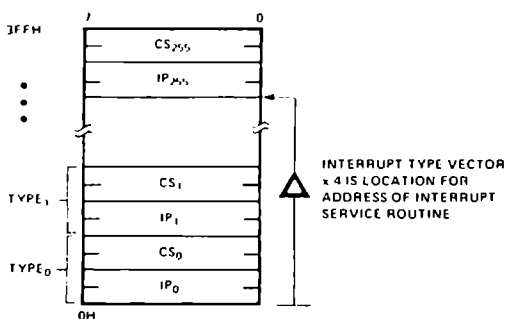
Reserved Interrupt Locations

Intel Corporation reserves the use of interrupts 0-31 (locations 00H through 7FH) for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

Interrupts 0 through 4 (00H-13H) currently have dedicated hardware functions as defined below.

Interrupt	Location	Function
0	00H-03H	Divide by zero
1	04H-07H	Single step
2	08H-0BH	Non-maskable interrupt
3	0CH-0FH	One-byte interrupt instruction
4	10H-13H	Interrupt on overflow

Interrupt Pointer Table



iAPX 86/88/186 Instruction Set Matrix

HI	Lo	0	1	2	3	4	5	6	7
0		ADD b:r/m	ADD w:r/m	ADD b:r/m	ADD w:r/m	ADD b:13	ADD w:13	PUSH ES	POP ES
1		ADC b:r/m	ADC w:r/m	ADC b:r/m	ADC w:r/m	ADC b:1	ADC w:1	PUSH SS	POP SS
2		AND b:r/m	AND w:r/m	AND b:r/m	AND w:r/m	AND b:1	AND w:1	SEG ES	DAA
3		XOR b:r/m	XOR w:r/m	XOR b:r/m	XOR w:r/m	XOR b:1	XOR w:1	SEG SS	AAA
4		INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5		PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6		PUSHA	POPA	BOUND r:r/m					
7		JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA
8		Immed b:r/m	Immed w:r/m	Immed b:r/m	Immed w:r/m	TEST b:r/m	TEST w:r/m	XCHG b:r/m	XCHG w:r/m
9		NOP	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI
A		MOV m ← AL	MOV m ← AX	MOV AL ← m	MOV AX ← m	MOVS b	MOVS w	CMPS b	CMPS w
B		MOV 1 ← AL	MOV 1 ← CL	MOV 1 ← DL	MOV 1 ← BL	MOV 1 ← AH	MOV 1 ← CH	MOV 1 ← DH	MOV 1 ← BH
C		Shift b:r/m,1	Shift w:r/m,1	RET (1 ← SP)	RET	LES	LDS	MOV b:r/m	MOV w:r/m
D		Shift b	Shift w	Shift b,v	Shift w,v	AAM	AAD		XLAT
E		LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCJZ	IN b	IN w	OUT b	OUT w
F		LOCK		REP	REP Z	HLT	CMC	Grp1 b:r/m	Grp1 w:r/m

where

mod r/m	000	001	010	011	100	101	110	111
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL/SAL	SHR	SHL/BAL	SAR
Grp1	TEST	—	NOT	NEG	MUL	IMUL	DIV	IDIV
Grp2	INC	DEC	CALL id	CALL 1id	JMP id	JMP 1id	PUSH	—

— 186 only instruction

iAPX 86/88/186 Instruction Set Matrix

HI	Lo							
	8	9	A	B	C	D	E	F
0	OR b/r/m	OR w/r/m	OR b/r/m	OR w/r/m	OR b/r/m	OR w/r/m	PUSH CS	I
1	SBB b/r/m	SBB w/r/m	SBB b/r/m	SBB w/r/m	SBB b/r/m	SBB w/r/m	PUSH DS	POP DS
2	SUB b/r/m	SUB w/r/m	SUB b/r/m	SUB w/r/m	SUB b/r/m	SUB w/r/m	SEG CS	DAS
3	CMP b/r/m	CMP w/r/m	CMP b/r/m	CMP w/r/m	CMP b/r/m	CMP w/r/m	SEG DS	AAS
4	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6	PUSH r/m	IMUL r/m, r/m	PUSH r/m	IMUL r/m, r/m	INS b	INS w	OUTS b	OUTS w
7	JS	JNS	JPE JPE	JNP JPO	JL JNGE	JNL JGE	JLE JNG	JNLE JG
8	MOV b/r/m	MOV w/r/m	MOV b/r/m	MOV w/r/m	MOV sr/r/m	LEA	MOV sr/r/m	POP r/m
9	CBW	CWD	CALL r/d	WAIT	PUSHF	POPF	SAHF	LAHF
A	TEST b/r/m	TEST w/r/m	STOS b	STOS w	LODS b	LODS w	SCAS b	SCAS w
B	MOV r ← AX	MOV r ← CX	MOV r ← DX	MOV r ← BX	MOV r ← SP	MOV r ← BP	MOV r ← SI	MOV r ← DI
C	ENTER r/m, b	LEAVE	RET r/m (r ← SP)	RET r	INT Type 3	INT (Any)	INTO	IRET
D	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	CALL r/d	JMP r/d	JMP r/d	JMP sr/d	IN r, b	IN r, w	OUT r, d	OUT r, w
F	CLC	STC	CLI	STI	CLO	STD	Grp 2 b/r/m	Grp 2 w/r/m

- b = byte operation
- d = direct
- f = from CPU reg
- i = immediate
- ia = immed to accum
- ib = immediate byte
- id = indirect
- is = immed byte sign ext
- iw = immediate word
- l = long io intersegment
- m = memory
- r = register
- r/m = EA is second byte
- sr = short intrasegment
- sr = segment register
- r = to CPU reg
- v = variable
- w = word operation
- z = zero

Clocks for MOD186 Operation

FUNCTION	FORMAT	186 Clock Cycles
DATA TRANSFER		
MOV = Move		
Reg. data to Reg. data Memory	1 0 0 0 1 0 0 = mod reg = mem	2:12
Reg. data memory to reg. data	1 0 0 0 1 0 1 = mod reg = mem	2:9
Reg. data to reg. data memory	1 1 0 0 0 1 1 = mod 000 = mem data data mem	12-13
Immediate to reg. data	1 0 1 1 = reg data data mem	3-4
Memory to accumulator	1 0 1 0 0 0 0 = address 000 high	9
Accumulator to memory	1 0 1 0 0 0 1 = address 000 high	8
Register memory to register register	1 0 0 0 1 1 1 0 = mod 0 reg = mem	2:9
Register register to register memory	1 0 0 0 1 1 0 0 = mod 0 reg = mem	2:11
PUSH = Push		
Memory	1 1 1 1 1 1 1 = mod 1 1 1 1 1 1 1	16
Register	0 1 1 1 1 1 1 = reg	10
Segment register	0 0 0 1 1 1 1 0 = reg = 0	9
Immediate	0 1 1 0 1 1 1 0 = data data data = 0	10
POPB = Pop B	0 1 1 0 0 0 0	36
POP = Pop		
Memory	1 0 0 0 1 1 1 1 = mod 000 = mem	20
Register	0 1 0 1 1 1 1 = reg	10
Segment register	0 0 0 1 1 1 1 1 = reg = 0	8
POPL = Pop AX	0 1 1 0 0 0 0 1	51
SCAS = Scan String		
Register memory to register	1 0 0 0 0 1 1 1 = mod reg = mem	4:17
Register to register	1 0 0 1 0 1 1 1 = reg	3
IN = Input from		
Input port	1 1 1 0 0 1 0 = port	10
Direct port	1 1 1 0 1 1 0 = port	8
OUT = Output to		
Output port	1 1 1 0 0 1 1 = port	9
Direct port	1 1 1 0 1 1 1 = port	7
SCAL = Scan String to AL	0 1 0 1 0 1 1	11
LEA = Load Effective Address	0 1 0 0 1 1 0 1 = mod reg = mem	8
LDS = Load String to DS	0 1 0 0 0 1 0 1 = mod reg = mem	18
LSS = Load String to SS	0 1 0 0 1 1 0 0 = mod reg = mem	18
LARL = Load Address to RAX	1 0 0 1 1 1 1 1	2
LAHL = Load Address to RAX	1 0 0 1 1 1 1 0	3
PUSHL = Push Long	1 0 0 1 1 1 0 0	9
POPL = Pop Long	1 0 0 1 1 1 0 1	8

shaded areas indicate instructions not available in iAPX 86/88 microsystems

Clocks for MOD186 Operation

FUNCTION	FORMAT	186 Clock Cycles
ARITHMETIC		
ADD = Add		
Reg. memory with reg. store to either	0 0 0 0 0 0 0 = mod reg. r/m	3-10
Immediate to reg. store memory	1 0 0 0 0 0 0 = mod 000 r/m data data size = 01	4-16
Immediate to accumulator	0 0 0 0 0 1 0 = data data size = 1	3-4
ADC = Add with carry		
Reg. memory with reg. store to either	0 0 0 1 0 0 0 = mod reg. r/m	3-10
Immediate to reg. store memory	1 0 0 0 0 0 0 = mod 010 r/m data data size = 01	4-16
Immediate to accumulator	0 0 0 1 0 1 0 = data data size = 1	3-4
INC = Increment		
Register memory	1 1 1 1 1 1 1 = mod 000 r/m	3-15
Register	0 1 0 0 1 reg	3
DEC = Decrement		
Reg. memory and reg. store to either	0 0 1 0 1 0 0 = mod reg. r/m	3-10
Immediate from reg. store memory	1 0 0 0 0 0 0 = mod 011 r/m data data size = 01	4-16
Immediate from accumulator	0 0 1 0 1 1 0 = data data size = 1	3-4
LEA = Load with address		
Reg. memory and reg. store to either	0 0 0 1 1 0 0 = mod reg. r/m	3-10
Immediate from reg. store memory	1 0 0 0 0 0 0 = mod 011 r/m data data size = 01	4-16
Immediate from accumulator	0 0 0 1 1 1 0 = data data size = 1	3-4
DEC = Decrement		
Register memory	1 1 1 1 1 1 1 = mod 011 r/m	3-15
Register	0 1 0 0 1 reg	3
CMP = Compare		
Register memory with reg. store	0 0 1 1 1 0 1 = mod reg. r/m	3-10
Register with register memory	0 0 1 1 1 0 0 = mod reg. r/m	3-10
Immediate with reg. store memory	1 0 0 0 0 0 0 = mod 011 r/m data data size = 01	3-10
Immediate with acc. store	0 0 1 1 1 1 0 = data data size = 1	3-4
NEG = Change sign	1 1 1 0 0 1 1 = mod 011 r/m	3
AAS = ASCII adjust for add	0 0 1 1 0 1 1 1	8
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1	4
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1	7
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1	4
MUL = Multiply (unsigned)	1 1 1 1 0 1 1 = mod 100 r/m	
Register Byte		26-28
Register Word		35-37
Memory Byte		32-34
Memory Word		41-43
IMUL = Integer multiply (signed)	1 1 1 1 0 1 0 = mod 101 r/m	
Register Byte		25-28
Register Word		34-37
Memory Byte		31-34
Memory Word		40-43
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 1 1 = mod reg. r/m data data size = 0	22-25/29-32
DIV = Divide (unsigned)	1 1 1 1 0 1 1 = mod 110 r/m	
Register Byte		29
Register Word		38
Memory Byte		35
Memory Word		44

Shaded areas indicate instructions not available in APX 86, 88 microsystems.

Clocks for MOD186 Operation

FUNCTION	FORMAT	186 Clock Cycles
ARITHMETIC (Continued)		
IDIV - Integer divide (signed)	1 1 1 0 1 1 = mod 111 m	44-52
Register: Byte		53-61
Register: Word		50-58
Memory: Byte		59-67
Memory: Word		19
AAM - ASCII adjust for multiply	1 1 0 1 0 1 0 0 0 0 0 1 0 1 0	15
AAS - ASCII adjust for divide	1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0	2
CWD - Convert word to double word	1 0 0 1 1 0 0 0	4
CWB - Convert word to double word	1 0 0 1 1 0 0 1	
LOGIC		
Scal: Rotate instructions		
Register: Memory by 1	1 1 0 1 0 0 0 = mod 111 m	2/15
Register: Memory by 01	1 1 0 1 0 0 1 = mod 111 m	5-n/17-n
Register: Memory by Count	1 1 0 0 0 0 0 = mod 111 m Count	5+n/17+n
	111 Instruction 0 0 0 ACR 0 0 1 ACP 0 1 0 BCR 0 1 1 BCP 1 0 0 CAC 1 0 1 CAP 1 1 1 SAR	
AND - AND		
Register: Memory and register to a register	0 0 1 0 0 0 0 = mod reg m	3/10
Immediate to register memory	1 0 0 0 0 0 0 = mod 100 m CAC CAC# = 1	4/16
Immediate to accumulator	0 0 1 0 0 1 0 = CAC CAC# = 1	3/4
TEST - AND function to flag; no result		
Register: Memory and register	1 0 0 0 0 1 0 = mod reg m	3/10
Immediate to register memory	1 1 1 1 0 1 1 = mod 000 m CAC CAC# = 1	4/10
Immediate to accumulator	1 0 1 0 1 0 0 = CAC CAC# = 1	3/4
OR - OR		
Register: Memory and register to a register	0 0 0 1 0 1 1 = mod reg m	3/10
Immediate to register memory	1 0 0 0 0 0 0 = mod 000 m CAC CAC# = 1	4/16
Immediate to accumulator	0 0 0 0 1 1 0 = CAC CAC# = 1	3/4
XOR - Exclusive OR		
Register: Memory and register to a register	0 0 1 1 0 0 0 = mod reg m	3/10
Immediate to register memory	1 0 0 0 0 0 0 = mod 000 m CAC CAC# = 1	4/16
Immediate to accumulator	0 0 1 1 0 1 0 = CAC CAC# = 1	3/4
Register: Register memory	1 1 1 1 0 1 1 = mod 010 m	3
STRING MANIPULATION		
MOVB - Move byte word	1 0 1 0 0 1 0 =	14
CMPS - Compare byte word	1 0 1 0 0 1 1 =	27
SCAS - Scan byte word	1 0 1 0 1 1 1 =	15
LODS - Load byte word to AL/AX	1 0 1 0 1 1 0 =	12
STOS - Store byte word to AL/AX	1 0 1 0 1 0 1 =	10
INS - Input byte word from I/O port	0 1 1 0 1 1 0 =	14
OUTS - Output byte word to I/O port	0 1 1 0 1 1 1 =	14

Shaded areas indicate instructions not available in APX 86/88 microsystems.

Clocks for MOD186 Operation

FUNCTION	FORMAT	186 Clock Cycles
STRING MANIPULATION (Continued)		
Reversed by Count in CA		
MOVE - Move string		8 + 8n
CMPS - Compare string		5 + 22n
SCAS - Scan string		5 + 15n
LODS - Load string		6 + 11n
STOS - Store string		6 + 9n
INS - Input string		8 + 8n
OUTS - Output string		8 + 8n
CONTROL TRANSFER		
CALL - Call		
Direct with segment		14
Register memory		13, 19
Indirect with segment		20
Direct with segment		
Indirect with segment		38
JMP - Unconditional jump		
Direct long		13
Direct with segment		13
Register memory, indirect with segment		11, 17
Indirect with segment		13
Indirect with segment		26
RET - Return from CALL		
Direct with segment		16
Register memory, indirect with segment		18
Indirect with segment		22
Indirect with segment, indirect with segment		25

Shaded areas indicate instructions not available in APX 86-88 microsystems

Clocks for MOD186 Operation

FUNCTION	FORMAT	180 Clock Cycles
CONTROL TRANSFER (Continued)		
JA <i>jump if above</i>	0 1 1 1 0 1 0 0	4-13
JAE <i>jump if above or equal</i>	0 1 1 1 1 0 0 0	4-13
JBE <i>jump if below or equal</i>	0 1 1 1 1 1 0 0	4-13
JNB <i>jump if not below</i>	0 1 1 1 0 0 1 0	4-13
JNBE <i>jump if not below or equal</i>	0 1 1 1 0 1 1 0	4-13
JNP <i>jump if not push</i>	0 1 1 1 1 0 1 0	4-13
JNPE <i>jump if not push or equal</i>	0 1 1 1 1 0 1 1	4-13
JNC <i>jump if not carry</i>	0 1 1 1 0 0 0 0	4-13
JNCB <i>jump if not carry below</i>	0 1 1 1 1 0 0 0	4-13
JNE <i>jump if not equal</i>	0 1 1 1 0 1 0 1	4-13
JNEB <i>jump if not equal below</i>	0 1 1 1 0 1 0 0	4-13
JNL <i>jump if not less</i>	0 1 1 1 1 0 1 1	4-13
JNLE <i>jump if not less or equal</i>	0 1 1 1 1 0 1 0	4-13
JNB <i>jump if not below</i>	0 1 1 1 0 0 1 1	4-13
JNBE <i>jump if not below or equal</i>	0 1 1 1 0 1 1 1	4-13
JNP <i>jump if not push</i>	0 1 1 1 1 0 1 1	4-13
JNPE <i>jump if not push or equal</i>	0 1 1 1 1 0 1 0	4-13
JNC <i>jump if not carry</i>	0 1 1 1 1 0 0 1	4-13
LOOP <i>loop</i>	1 1 1 1 0 0 1 0	4-13
LOOPZ <i>loop if zero</i>	1 1 1 1 0 0 0 1	5-15
LOOPNZ <i>loop if not zero</i>	1 1 1 1 0 0 0 0	6-16
LOOPJL <i>loop if jump less</i>	1 1 1 1 0 0 0 1	6-16
JCL <i>jump if carry</i>	1 1 1 1 0 0 1 1	16
		5
ENTER <i>Enter Procedure</i>		
L = 0	1 1 0 0 1 0 0 0	15
L = 1		25
L = 1		22 + 10(n - 1)
LEAVE <i>Leave Procedure</i>	1 1 0 0 1 0 0 1	8
INT <i>Interrupt</i>		
Type 0 <i>INT0</i>	1 1 0 0 1 1 0 1	47
Type 1	1 1 0 0 1 1 0 0	45
INTB <i>Interrupt on bus</i>	1 1 0 0 1 1 1 0	48-4
INVT <i>Interrupt Vector</i>		
	1 1 0 0 1 1 1 1	28
OWORD <i>Overflow out of range</i>		
	0 1 1 0 0 0 1 0	33-35

Shaded areas indicate instructions not available in APX 86/88 microsystems



3065 Bowers Avenue, Santa Clara, California 95051
(408) 987-8080

Printed in U.S.A.

A1123/283/10k/J/KH